Was the Jury Wrong About Toyota’s Software?
How Questionable Testimony on Embedded Software Tipped the Scales

David M. Cummings
dcummins@kellytechnologygroup.com

This article appears in the July 2017 issue of IEEE Consumer Electronics Magazine, pages 103-107 [DOI: 10.1109/MCE.2017.2684939].

© 2017 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

1 Introduction

(This is a modified version of an article appearing in the December 2016 issue of IEEE Technology & Society Magazine, pages 76-84 [DOI: 10.1109/MTS.2016.2618681]. In this article, references are made to the analysis from the original IEEE T&S article where appropriate.)

Product liability trials involving embedded software, such as those in which an automobile driver alleges that faulty embedded software was responsible for an accident, require the same standard of proof as other civil trials. The plaintiff must convince a jury that the defendant (for example, the automobile manufacturer) more likely than not is responsible for the accident [1]-[4]. This includes showing that the defendant failed to fulfill their duty, and that the defendant’s failure was the cause of the accident (causation) [5]-[12]. The defendant may offer evidence to counter the plaintiff’s arguments. For example, if the plaintiff’s technical expert testifies that in his/her opinion it is more likely than not that a specific problem in the defendant’s embedded software caused the accident, the defendant might identify flaws in the analysis. As will be shown, when a non-technical jury in such a trial is confronted with complex software issues, the verdict they deliver may not be supported by the facts. The jury can get it wrong.

It is well known that nearly all non-trivial software has bugs [13]-[15]. Furthermore, because there are virtually an infinite number of different ways of solving a non-trivial problem using software, one can often find many opportunities for criticizing software quality, sometimes using criteria that are highly subjective. As a result, the plaintiffs in a software trial can be expected to attack the defendant’s software by looking for bugs and criticizing the software’s quality. If they can find bugs, and show that more likely than not those bugs caused the accident, then they can establish causation. They can also use purported measures of software quality, which may be subjective, to argue that the defendant failed to fulfill their duty to provide software of sufficient quality.
Was the Jury Wrong About Toyota’s Software?: How questionable testimony on embedded software tipped the scales
David M. Cummings

This can present unique challenges in software cases. The plaintiffs’ expert can bombard a non-technical jury with criticisms of the defendant’s software that sound convincing, but which may or may not be justified because of the subjective nature of many such criticisms. He/she can then conclude that because the defendant’s software is substandard, it is more likely than not that it was responsible for the accident. The problem with this line of argument is that even if bugs or design flaws are found in the defendant’s software, this does not mean the software caused the accident. The plaintiffs may not have fulfilled their responsibility to show a connection between the potential bugs and the accident. The plaintiffs’ expert may convince him/herself that they have found the cause of the accident, when all they have really found are potential bugs unrelated to the accident. Alternatively, the plaintiffs’ expert may realize that they have not found a credible connection between the potential bugs and the accident, but they are willing to tell the jury that more likely than not those bugs are responsible for the accident. (That would be a violation of ethical standards including IEEE ethical standards [16] and ACM ethical standards [17].) Either way, the jurors are told by an expert that problems were found in the software, and in the expert’s opinion it is more likely than not that those problems caused the accident.

In order to illuminate these issues, we will look at a specific case, the 2013 Oklahoma trial in which a jury determined that Toyota’s engine control software was to blame for reported unintended acceleration that ended in a fatal accident. We are fortunate in that one of the plaintiffs’ software experts made his trial testimony and slides available to the public (see links below). This gives those of us in the engineering community a rare opportunity to get a first-hand look at the technical arguments that were presented. The expert’s post-trial commentary included the following:

...The second round began with an over 750 page formal written expert report by me in April 2013 and culminated this week in an Oklahoma jury’s decision that the multiple defects in Toyota’s engine software directly caused a September 2007 single vehicle crash that injured the driver and killed her passenger. ...In a nutshell, the team led by Barr Group found what the NASA team sought but couldn’t find: “a systematic software malfunction in the Main CPU that opens the throttle without operator action and continues to properly control fuel injection and ignition” that is not reliably detected by any fail-safe. ...Now it’s your turn to judge for yourself. Though I don’t think you can find my expert report outside the Court system, here are links to the trial transcript of my expert testimony to the Oklahoma jury and a (redacted) copy of the slides I shared with the jury in Bookout, et.al. v. Toyota. [18]

As the expert suggested, I examined his trial materials and judged for myself. I had a particular interest in this subject, having previously written an opinion piece for the Los Angeles Times about Toyota’s investigation into the cause of reported unintended acceleration, and the possibility that software was to blame [19]. Based on the statements made by the expert, including those above, as well as favorable articles in various publications describing his trial testimony, I expected to find a convincing case built on compelling evidence that there was indeed “a systematic software malfunction in the Main CPU” that directly caused the automobile accident. Unfortunately, I found instead a disappointingly flawed theory of causation that is not credible. (The National Highway Traffic Safety Administration [NHTSA] apparently agrees with this assessment [20].)
Was the Jury Wrong About Toyota's Software?: How questionable testimony on embedded software tipped the scales
David M. Cummings

Thus, this trial serves as an informative case study demonstrating how a jury can be led to an apparently incorrect conclusion about software based on technical arguments that sound convincing, but which do not withstand technical scrutiny. And as we become increasingly reliant on embedded software in our daily lives, for example, with the advent of self-driving cars, trucks, and automotive “autopilot” features, our legal system is likely to face this issue with increasing frequency.

Since my review of the trial materials, my company has been retained by several automobile manufacturers. Confidentiality agreements prevent me from revealing details about those engagements.

2 Background

I will refer below to the expert’s trial testimony and slides, which can be found at:

- http://www.safetyresearch.net/Library/Bookout_v_Toyota_Barr_REDACTED.pdf (“Testimony”)
- http://www.safetyresearch.net/Library/BarrSlides_FINAL_SCRUBBED.pdf (“Slides”)

The expert concluded that the accident was caused by the death of a critical task within the engine control software, which he referred to as “Task X.” He testified:

So to a reasonable degree of engineering certainty, it's my opinion that it was more likely than not, a task X death, possibly in combination with other tasks that occurred that day, causing a loss of throttle control and in [sic] inability to stop the vehicle's full momentum because of the vacuum loss. [Testimony, PDF page 192.]

See also Testimony, PDF page 187, and Slide 54.

This conclusion does not appear to be supported by the evidence presented at trial.

Before examining the expert’s conclusion in detail, some background information about Task X, derived from the above trial materials, is required. Also required is background information, derived from the trial materials, regarding a fail-safe called the “Brake Echo Check,” which is a key element of the accident theory involving the death of Task X.

Task X is a periodic task that executes multiple times per second on the engine control processor (the main CPU). One of its many responsibilities is to determine the correct throttle angle setting (how far open the throttle should be) based on how hard the driver is pressing on the accelerator pedal (as well as other factors). Therefore, every \( n \) milliseconds\(^1 \) Task X wakes up and, among other things, determines the current accelerator pedal position and sets a throttle angle variable accordingly. The throttle angle variable is then used by another part of the software to set the throttle to the angle specified in that variable.

All the tasks running on the main CPU are managed by a multitasking operating system. The operating system maintains a data structure containing one bit per task. If the bit is set, the task is alive and is

\[\text{---}\]

\(^1\) The Slides suggest that Task X runs every 8 milliseconds [Slide 35].

3
subject to task scheduling. If the bit is clear, the task is not running and it will not be scheduled for execution. According to the expert, this data structure (as well as the throttle angle variable) was not protected by software techniques such as mirroring, or by hardware techniques such as error detection and correction. Therefore, if this data structure became corrupted due to a software bug or a single event upset, the corruption would not be detected or corrected.

There is a second processor called the monitor CPU that executes the Brake Echo Check fail-safe software. The Brake Echo Check is designed to behave as follows: If Task X died, and if the driver then stepped on the brake or released the brake, then about 200 milliseconds later the Brake Echo Check on the monitor CPU would detect an inconsistency resulting from the death of Task X on the main CPU, and would force the throttle to idle. About 3 seconds later it would stall the engine. When the throttle is at idle, braking will successfully stop the vehicle.

3 Theory Regarding the Death of Task X

According to the accident theory presented to the jury, the following 3 things had to happen together just prior to the accident:

A. The bit corresponding to Task X in the operating system data structure was somehow flipped from one to zero, resulting in the death of Task X.

B. At the time of this bit flip, the throttle angle variable maintained by Task X contained a large value, corresponding to an open throttle. Because Task X never ran again, the throttle angle variable was stuck at this value, and the throttle remained open.

C. The Brake Echo Check did not work for some reason. When the driver stepped on the brake, the Brake Echo Check did not correctly detect the inconsistency due to the death of Task X and force the throttle to idle. Because the throttle remained open, the driver was unable to stop the vehicle by braking.

Let’s examine each of these:

- A is merely hypothetical. No evidence was presented that this bit was flipped prior to the accident, nor was any specific bug in the software identified that caused this bit to flip. Instead, the expert identified what he said were a number of problems in the software that could possibly cause an unspecified memory corruption under some circumstances, which he speculated might possibly include the corruption of this bit.

- B appears to be inconsistent with the facts of the accident. The driver was slowing the vehicle on an exit ramp when the expert theorizes Task X died, and therefore the throttle would likely have been at idle.

2 Descriptions of the intended operation of the Brake Echo Check can be found in Testimony, PDF pages 135, 174, 225-226, and 245.

3 For example, see Testimony, PDF pages 70-72 and 90-96, and Slides 19, 21, and 37. See also the related analysis in the IEEE T&S article.

4 For example, see Testimony, PDF pages 223-224. See also the related analysis in the IEEE T&S article.
C is merely hypothetical. Even if the bit flipped, and even if the throttle was open when that happened, there is no evidence that the Brake Echo Check, executing on a different processor, would not have worked correctly to force the throttle to idle. In fact, the expert did not even speculate at trial why the Brake Echo Check might possibly fail under any circumstances. He simply asserted that it was unreliable without providing any reasons. (He said that his expert report contained reasons, but he couldn’t recall any of them [Testimony, PDF page 147].) Moreover, all of the Brake Echo Check testing he described showed it working exactly as designed after the death of Task X. 

Thus, this theory is not credible as the likely explanation for the accident for at least the following reasons:

a. It requires the nearly simultaneous occurrence of 2 hypothetical failures, A and C.

b. For hypothetical failure A, no evidence was provided that it occurred at the time of the accident, or under any circumstances. The expert merely speculated that it might possibly occur under some circumstances due to problems he claimed to have identified in the software. No connection was established between any of those claimed problems and the specific bit in question.

c. For hypothetical failure C, no evidence was provided that it occurred at the time of the accident, or under any circumstances. In fact, at trial the expert merely proposed that it occurred without even speculating why.

d. Also for C, all of the testing the expert described showed the Brake Echo Check working exactly as designed.

e. A and C are independent. Hypothetical failure A is associated with software on a different processor than the software associated with hypothetical failure C. What’s more, no argument was made that there was a dependence between these two hypothetical failures. The probability of two independent low-probability failures occurring together is much lower (multiplicatively) than the probability of either one occurring alone. Because no evidence was presented that either A or C occurred under any circumstances, it is reasonable to treat each of these as very low probability failures. The probability of both occurring together, then, would be expected to be extremely low (all the more so because not even a speculative reason was offered as to why C might ever occur).

f. In addition, the theory requires a third item, B. But B appears to be contradicted by the facts of the accident. That makes the theory, which already lacks credibility due to the extremely low probability of A and C ever occurring together, even less credible.

Can we completely rule out the theoretical possibility that under some circumstances, A, B, and C could conceivably occur together, as improbable as that may be? Maybe not. But the mere possibility that this could theoretically happen is not enough to conclude “to a reasonable degree of engineering certainty”

---

5 For example, see Testimony, PDF pages 225, 233-234.
6 For hypothetical failure A, a single event upset was also suggested as a potential cause [Testimony, PDF page 72]. Single event upsets are very low probability failures. The probability of a single event upset affecting a specific bit is even lower.
that this is the likely explanation for this accident. (It is all the more improbable because the facts of the accident suggest that B did not occur. Furthermore, no reason was given why C would ever occur under any circumstances.)

The expert emphasized that the Brake Echo Check fail-safe only acts if the driver steps on the brake or releases the brake. He suggested that because of this, if the driver’s foot was already on the brake when Task X supposedly died, the Brake Echo Check fail-safe would not act to force the throttle to idle. [Testimony, PDF pages 89, 135-136, 233-234.] As shown in the IEEE T&S article, this is irrelevant. If the driver’s foot was on the brake when Task X died, then there is no need for the Brake Echo Check fail-safe, because the throttle would already be at idle when Task X died, and braking would stop the vehicle.

4 Alternative Theory Regarding the Death of Task X

The jury was given an alternative theory based on the assumption that, in addition to the hypothetical bit flip causing the death of Task X, a second hypothetical memory corruption overwrote the throttle angle variable with a large value. As shown in the IEEE T&S article, this alternative theory is not credible as the likely explanation for the accident.

5 Related Testimony and Slides

There are a number of trial slides that apparently are intended to provide evidence for the expert’s accident theories, but in fact they do not. Examples are Slides 20, 53, and 49. These slides and related testimony are discussed in the IEEE T&S article.

6 Criticisms of Software Quality

The expert presented many slides criticizing Toyota’s software quality. See, for example, Slides 11, 14, 23-25, 27-33, 35, 38, 39, 41, 43-48, 50-52.

A second expert also offered testimony criticizing Toyota’s software quality [21]-[23], although he did not examine Toyota’s source code [24]-[25]. That expert did not provide an opinion of his own regarding causation, nor did he provide an opinion about the first expert’s causation theories [26].

A number of the criticisms of Toyota’s software quality from the two experts appear to be highly subjective, although they were presented to the non-technical jury as objective fact. For example, both experts criticized the number of global variables they said were found in Toyota’s one million lines of source code\(^7\), telling the jury that this was one of several indicators that the software was of poor quality.

The second expert told the jury that he did not need to see the source code to determine if there is a problem related to global variables (and in fact he did not see Toyota’s source code). All he needed to

\(^7\) See Testimony, PDF pages 28 and 180 for the number of lines comprising Toyota’s source code.
know is the count of global variables. He said the academic standard for this number is zero, and that “global variables are evil”:

A My next opinion is Toyota’s source code is of poor quality. And as you know, I haven't seen the source code myself ... Even at a high level, there is some tell-tale signs that you don't need to look at the individual lines of code to know there are some severe problems here.

One of them is 10,000 global variables. If you talk to a safety person, and that number is above 100. Even if it is 100, they will right there say, You know, that's it. There is no way this can be safe.

Q Isn't the actual academic standard there should be zero global variables?
A That academic standard is there should be zero. In fact, I have a chapter in my book called Global Variables Are Evil, and that was written in 2009.

Q And Toyota's system has 10,000 global variables?
A About 10,000. The number depends how you count. We will get to that, but that is the ballpark. Yes. [27]

Well, I know that the right answer academically is zero. And in practice, five, ten, okay, fine. 10,000, no, we’re done. It is not safe, and I don't need to see all 10,000 global variables to know that that is a problem. [28]

According to the second expert’s testimony, as long as the number of global variables in Toyota’s one million lines of code exceeds a small value (“five, ten”), then there is no need to actually look at the code to understand why the decision to implement them as global variables was made. If the number exceeds this small value, then he knows there is a problem, regardless of how or why the global variables are used.

However, as I will now show, this expert’s own academic code violates his standards for the use of global variables. His academic code exhibits the same use of global variables that he told the jury was an indicator of poor quality software.

On his university website, the expert advertises a software project that he led called Ballista: http://users.ece.cmu.edu/~koopman/ballista/. He provides the Ballista source code for download at: http://users.ece.cmu.edu/~koopman/ballista/ostest/ostest_download.html.

That source code includes 6 C files, 19 .h files, and 21 C++ files, among others. These files comprise a total of 8,552 lines of code. In those 8,552 lines, which is less than 1% of the number of lines in Toyota's software, there are a total of at least 68 global variables.

---

8 These file counts refer to what appear to be manually created C/C++ source files. There are also a large number of source files within a directory called “templates” that are apparently automatically generated from a tool, as well as a large number of Perl files within a directory called “perllib.” Those files are not included in this analysis.

9 See ballista.cpp lines 59-88 (7 globals); callGen.cpp line 26 (1 global); callGen_standAlone.cpp line 26 (1 global); serverCommunication.cpp lines 79-83 (5 globals); blexer.c lines 25-26 (2 globals); bparser.cpp lines 39-105 (48 globals); tempname.c lines 22-25 (4 globals).
Was the Jury Wrong About Toyota’s Software?: How questionable testimony on embedded software tipped the scales
David M. Cummings

Since Ballista is a software project created in an academic environment by the expert and his research group, one would expect that the academic standard that he presented to the jury, which is zero global variables, should apply. But he clearly did not apply that standard to his own software. Furthermore, the number of global variables in his 8,552 lines of code greatly exceeds his threshold of acceptability even for non-academic code, which is “five, ten” global variables in Toyota’s one million lines of code according to his testimony. Thus, the standards regarding global variables that he presented to the jury are apparently not the standards that he applies to his own code.

Furthermore, there are comments in the Ballista source code that describe the rationale for making some of the variables global. For example:

```
//don't want to pay the overhead of creating
//and deleting many times
```

and:

```
//These are global so they can be destroyed properly when aborting
//Since only one of each is created during the entire program, having them
//as global should not be a problem.  
```

These comments reflect just a few of the many possible considerations that could lead to a decision to make variables global. Such considerations were not acknowledged by either expert during the trial testimony. Rather, the impression conveyed to the non-technical jury was that the numbers presented at trial (no more than “five, ten” global variables in one million lines of code) are hard and fast, and if those numbers are exceeded then the software is of low quality, period. This is not an accurate reflection of real-world software development, as the second expert’s own code shows, but nonetheless this is what the jury was told.

This is just one of several examples of the subjective nature of the criticisms of Toyota’s software quality presented to the jury by both experts. Other examples can be found in a follow-on paper to the IEEE T&S article.

7 Causation not Demonstrated

The jury was told that “to a reasonable degree of engineering certainty, it was more likely than not” that Task X death caused the accident. Two theories were presented to support this conclusion, but neither theory was credible. Nonetheless, the jury accepted the conclusion.

The message provided to the jury by the plaintiffs was twofold:

- The defendant’s software was poorly designed and full of bugs.
- These bugs likely caused the accident.

---

These 2 comments are found in `ballista.cpp`. 
As with all non-trivial software, there likely were bugs in the defendant’s software. Perhaps the plaintiffs found some of them. But this does not mean that the accident was caused by any of the purported bugs. Causation was not demonstrated.

Due to the nature of software, it seems likely that this same scenario will play out in future embedded software trials. A key to ensuring that an informed decision will be reached by the jury in such a trial is for both plaintiffs and defendants to adequately address the second bullet point above, causation. Otherwise, the jury can be convinced that causation has been demonstrated when in fact all that has occurred is that the quality of the defendant’s software has been criticized, perhaps legitimately and perhaps not. This appears to be what happened in this trial.

8 References


Was the Jury Wrong About Toyota’s Software?: How questionable testimony on embedded software tipped the scales
David M. Cummings


[26] Transcript Of Afternoon Trial Proceedings, October 13, 2013, [22], pages 80-82, 111.

[27] Transcript Of Morning Trial Proceedings, October 13, 2013, [21], pages 33-34.